Technical Report ITL-96-6
July 1996

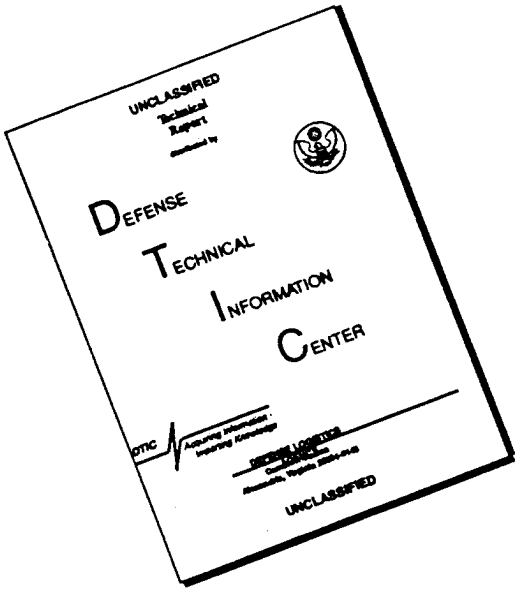
US Army Corps of Engineers
Waterways Experiment Station

APEX CM-VC and Reverse Engineering

by Pamela M. Woof, University of Memphis

Prepared for Headquarters, U.S. Army Corps of Engineers

DISCLAIMER NOTICE



THIS DOCUMENT IS BEST QUALITY AVAILABLE. THE COPY FURNISHED TO DTIC CONTAINED A SIGNIFICANT NUMBER OF PAGES WHICH DO NOT REPRODUCE LEGIBLY.

Technical Report ITL-96-6
July 1996

APEX CM-VC and Reverse Engineering

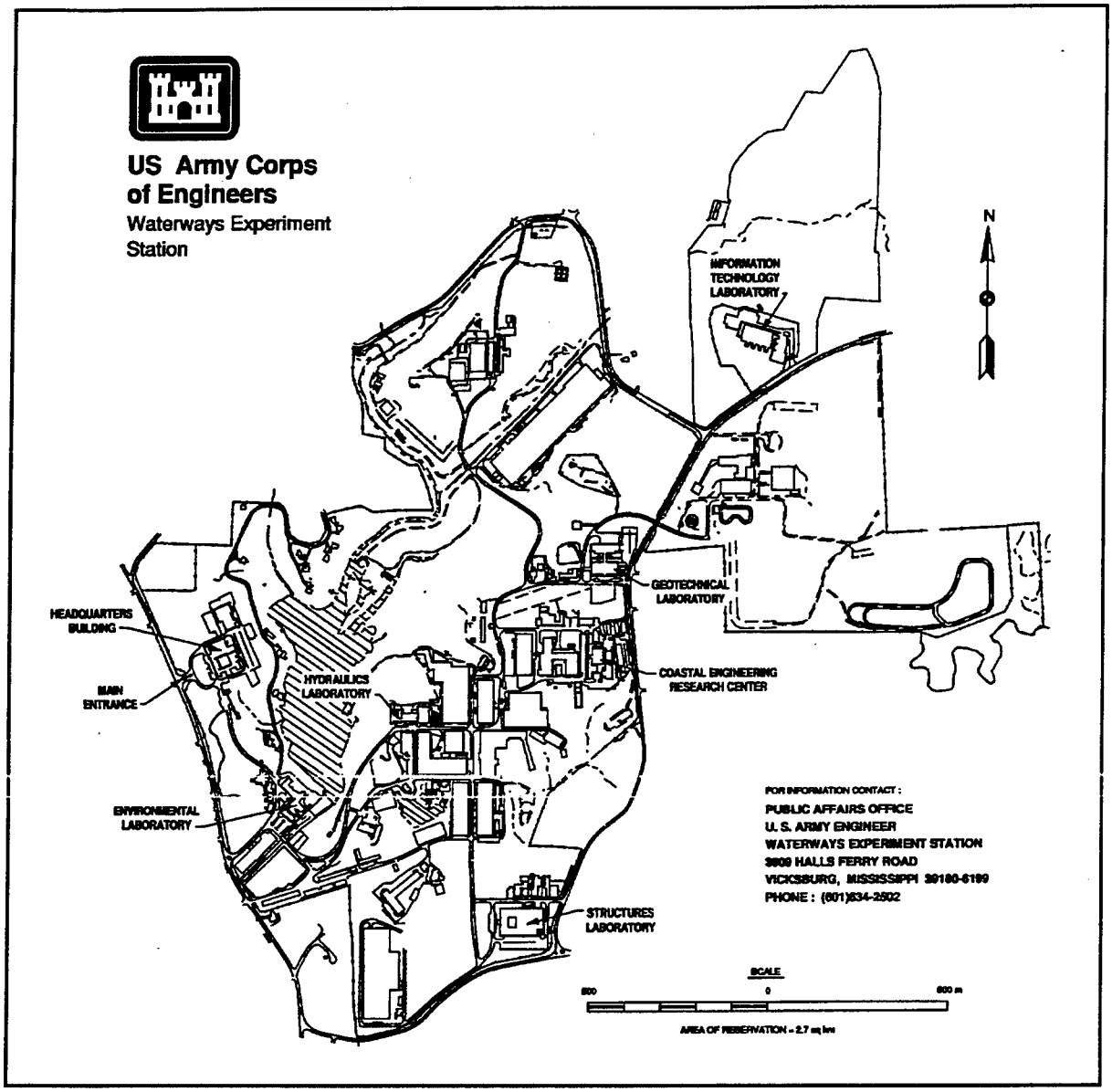
by Pamela M. Woof
University of Memphis
Memphis, TN 38152

Final report

Approved for public release; distribution is unlimited

Prepared for U.S. Army Corps of Engineers
Washington, DC 20314-1000

Monitored by U.S. Army Engineer Waterways Experiment Station
3909 Halls Ferry Road, Vicksburg, MS 39180-6199



Waterways Experiment Station Cataloging-in-Publication Data

Woof, Pamela M.
APEX CM-VC and reverse engineering / by Pamela M. Woof ; prepared for U.S. Army Corps of Engineers ; monitored by U.S. Army Engineer Waterways Experiment Station.
26 p. : ill. ; 28 cm. — (Technical report ; ITL-96-6)
1. APEX (Computer program) 2. Computer software — Development. 3. Ada (Computer program language) 4. Software configuration management. I. United States. Army. Corps of Engineers. II. U.S. Army Engineer Waterways Experiment Station. III. Information Technology Laboratory (U.S. Army Engineer Waterways Experiment Station) IV. Title. V. Series: Technical report (U.S. Army Engineer Waterways Experiment Station) ; ITL-96-6.
TA7 W34 no.ITL-96-6

CONTENTS

PREFACE

The development and fielding of large software systems has reached a level of complexity that requires automated methods for keeping track not only of the development effort, but also, of the various versions of the system deployed to users. These activities are characterized by the terms "configuration management" and "version control." There are stand alone products available to perform these tasks separately or together. It seems likely that such software will be more effective when integrated with the original development environment rather than used in a stand alone fashion. This report considers the integrated configuration management and version control available in the most complete integrated software development system for Ada. It also briefly examines a graphical design tool capability, reverse engineering, that is of value in maintaining and updating legacy code.

The Apex programming environment (from Rational Software Corporation) is a complete development environment for producing programs in the Ada language. It is typically used in the development of large systems that must meet strict reliability requirements among other things, and that are expected to be long lived. This report is one of several documenting investigations into various aspects of its use.

Rose/Ada is a graphical design tool used either in conjunction with Apex or as a standalone product. The present interest, of course, is in its use as an integral part of the Apex system. It is a layered product (also from Rational Software Corporation) that is executable from the tool menu within Apex.

This document is for persons looking at the Apex system for the first time, either for evaluation or actual use of the system. It is not intended as a general introduction to Apex, however, but as an introduction to only two of its many capabilities. The goal of this document is to introduce the reader to some of the essential terminology and concepts used in Apex and Rose/Ada. This report is not intended to replace the online tutorials or other documentation. (The new user, particularly, is strongly advised to work through the tutorials in full.)

The reader is first introduced to the Apex environment. The interface is briefly discussed. The Apex concept of a view is discussed at length. Views encourage intelligent separation of large projects into related groups, and the view structure defines the interaction between the groups and the change and update restrictions within a group. The procedure to import foreign text files into Apex views is described, and sample results are included. Configuration management and version control within Apex is explained, and its impact on Apex file management is discussed.

The discussion of Rose/Ada is limited to the diagrams produced through the Apex "reverse engineer" function. The procedure to reverse engineer an existing system is discussed. As an example, an existing system is used to generate diagrams, and the diagrams and their evolution is included.

This report was written for the U.S. Army Engineer Waterways Experiment Station (WES) by Ms. Pamela M. Woof, University of Memphis, under Contract No. DACA39-94-K-0052, "Software Engineering Improvement with Ada." This study was performed under the supervision of Dr. Windell F. Ingram, Chief, Computer Science Division, Information Technology Laboratory (ITL), WES, and Dr. N. Radhakrishnan, Director, ITL.

During publication of this report, Director of WES was Dr. Robert W. Whalin. Commander was COL Bruce K. Howard, EN.

Executive Summary

Apex is an Ada development environment. This environment contains tools for developing large software projects, including file management and control tools. Apex is designed to be a multi-user environment, and includes file safeguards to lend stability to the development. The environment also allows several versions of any Ada unit to be held in reserve, to be used in the future.

Apex divides the work area into subsystems that contain views. Views are the file access mechanisms that coordinate the Ada files. All development work is performed within views. There are several different kinds of views with differing levels of file management options and restrictions.

The configuration management and version control (CMVC) utility is used to limit file access within view constraints. CMVC changes the system attributes of the files under its control so that only Apex may make any modifications to the files. In addition, CMVC guarantees that each file may only be changed or updated by one user or process at a time. CMVC also keeps track of old versions of each file and makes these versions available for certain functions.

Rational Rose/Ada is an object oriented design diagramming tool for use with Ada programs. Within Apex, Rose/Ada offers the option to build a module diagram for the Ada system being developed. This automatic generation of diagrams can be very helpful in the review of a system.

This report is not intended to serve as an introduction to all of Apex. Included are discussions of the view as a work space, importing foreign text files, importing between views, configuration management and version control, and the interaction between Apex and Rational Rose/Ada. The discussion of Rational Rose/Ada will be limited to the diagrams that are produced through the Apex reverse engineer function.

Apex Introduction

Apex is an Ada development environment. It contains all the standard tools for development: text editor, compiler, linker/binder, and debugger. In addition, Apex also includes facilities for file management: creating and maintaining directories and files.

This environment is designed to develop large software projects and support a multi-user environment. It includes options to safeguard files by making them accessible to only one user or process at a time, and options to restrict changes to sets of files.

The Apex interface consists of multiple windows and is mostly menu driven. For each window, the most used commands are located on a tool bar. When a command requires additional information, a dialogue window is provided. In this window, Apex prompts for the needed information, and many of the parameters may be selected from pick lists. All windows include the option to exit Apex, and Apex cleans up all the windows it has generated before it exits. A well designed tutorial and a help facility are available from any window.

The text editor provides Ada-specific editing, pretty-printing, and Ada specific analysis. The pretty-printing maintains a consistent style between development units. The Ada specific analysis includes some automatic correction of syntactic and semantic errors, and includes the option to traverse to the page in the online *Reference Manual for the Ada Programming Language* that is relative to the error. In addition, all coding may be accomplished from the menu in the text editor, and errors are highlighted for immediate correction.

Apex divides the work area into subsystems. These subsystems exist physically as directories, and act as Ada libraries. They are used to group related Ada files and to coordinate object files. Each subsystem contains a set of views.

The View as a Work Space

All development is performed within views. A view is a file access mechanism within a subsystem. It exists as a directory within the subsystem directory. The view keeps track of the active version of each of its files, and makes other versions available for certain functions. Characteristics of the view as a whole may be controlled within Apex, such as final compilation state and availability for modification.

A view contains a set of related Ada files. A subsystem may contain several views of different types without conflict. These views may contain the same units at different stages of development, but units in one view cannot depend on units in another view in the same subsystem. The multiple views are intended to be used for differing arrangements of versions of the same set of units.

Two types of views exist in Apex: working and release. Both are groupings of related files. Development occurs in working views, and release views are copies of working views that have restrictions on modification. Both working and release views have the option of being designated Interface Only when created; only specification units will be accepted into the view.

The working view has no restrictions on file access other than the standard CMVC control. Files may be accessed and modified, and the active version of any file may be changed. Release views have restrictions on the modification of included files. When files are imported into any view, they are automatically updated to the goal state of that view.

Goal States

Each view has a goal state. The goal state is the level of investigation that Apex is to perform on the code: archived, source, installed, coded or linked. When copying units from another view into the current view, the unit is updated to the goal state of the current view.

When a new view is created, the default goal state is coded. When a view is copied, the copy interface includes the option to change the goal state of the copy.

Archived code is not examined at all by Apex. This is usually code that is not ready to be compiled.

Source code is parsed and syntactically completed. Ending punctuation such as right parentheses, semicolons, closing quotation marks, and matching identifiers in the end statements of loops, blocks, subprogram bodies, and packages are inserted, if missing. Parsing may also pretty print the file by adjusting indentation level, line breaks, spacing, and capitalization.

Installed code is checked for semantic consistency. Type compatibility, correct subprogram parameters, and declaration of all objects are checked. The code is also parsed and syntactically completed.

Coded code has been parsed, syntactically completed, checked for semantic consistency, and compiled into the subsystem. Coding creates the object code related to each unit. Apex recognizes dependencies between units, and will make sure that all units required to successfully code the unit in question are already included in the subsystem object files. If any required units are not already coded into the subsystem, Apex will search for them and code them, if possible. If this is not possible, an error message is returned.

Linked status means that an executable has been generated based on this component after it has been coded. The linking command automatically searches for all required units and codes them into the subsystem, if they are not already coded. If the required units cannot be found or cannot be coded successfully, an error message is returned.

Release Views

Views may be either working or release. Working views contain Ada files that are still evolving. When these files reach a stable level, they may be copied into a release view. The release views are used to limit changes to the enclosed files, and have progressive levels of security.

Release views may be development, stable, or frozen. They are usually created by making a copy of an existing working view and using the copy options to force the new copy to be a release

view with the desired characteristics.

Development release views are often used as integration areas, and promote stability by prohibiting in-place editing. Units may be imported from other release views, and units may be updated by accepting changes. All compilation commands are available, but in-place file changes or deletions are not allowed. A development release may be changed to stable or frozen.

Stable release views are more restrictive than development views. They serve as an interim step on the way to frozen release views. The only way to adjust the view's contents is to import from another release view. All compilation commands are available. A stable release may be changed to frozen or back to development if necessary.

Frozen release views are the most restrictive. No compilation commands or CMVC commands are available. All directories and files are read-only. The only operation available in frozen views is the importation of other frozen views. Once a view is set to frozen, it cannot be changed back to stable or development. A frozen release view may be copied, and the attributes of the copy may be changed to stable or development with copy options. If the copy option is not changed, the copy will also be frozen and unchangeable.

Importing Foreign Text Files

Apex has a built in facility for translating Ada text files from other applications into files that can be accepted into an existing view. This allows projects started in other environments to be moved into the Apex environment with few difficulties.

One command from the Directory Viewer main menu, Import Text Files, is all that is needed. The foreign files to be imported are selected and directed to an existing view, and Apex does the rest. Apex will copy the text file into a new file with an Ada suffix (*.1.ada* or *.2.ada*), and will analyze the code according to the goal state of the view, usually by attempting to compile each file in the order of their interdependencies. The best way to limit confusion is to have all of the text files to be imported in one directory, and make sure that the target view exists. This way, there is no question which files are text files and which files are Apex Ada files.

Apex does not recognize stub bodies, and will not be able to recognize garbage characters that may be inserted at the end of the file by a file transfer protocol. Apex will recognize a file containing several separates, and will accordingly break the file into individual Ada units in their own file according to procedure or function name. Apex will return a list of the files that could not be imported into the chosen view, and these files should be checked for stub bodies and garbage characters.

For example, the one file that contained the following list of procedures was imported into Apex:

```
separate (Db_Gate)
procedure Move_To_Directory is . . .
procedure Create_Database is . . .
procedure Open_Database is . . .
```

procedure Close_Database is . . .
function Database_Exists is . . .
function Is_Empty is . . .
procedure Put_Installation_Record is . . .
procedure Put_Project_Record is . . .
procedure Put_Narrative is . . .
procedure Update_Installation_Record is . . .
procedure Update_Project_Record is . . .
procedure Update_Narrative is . . .
procedure Get_Installation_Record is . . .
procedure Get_Project_Record is . . .
procedure Get_Narrative is . . .
procedure Set_Status is . . .

This file was broken into the following list of files that were accepted into the Apex system:

db_gate.close_database.2.ada
db_gate.create_database.2.ada
db_gate.database_exists.2.ada
db_gate.get_installation_record.2.ada
db_gate.get_narrative.2.ada
db_gate.get_project_record.2.ada
db_gate.is_empty.2.ada
db_gate.move_to_directory.2.ada
db_gate.open_database.2.ada
db_gate.put_installation_record.2.ada
db_gate.put_narrative.2.ada
db_gate.put_project_record.2.ada
db_gate.set_status.2.ada
db_gate.update_installation_record.2.ada
db_gate.update_narrative.2.ada
db_gate.update_project_record.2.ada

Importing Between Views

In a large software project, it is convenient to divide the project into smaller pieces. Each of these pieces may be contained within their own subsystem, so Apex provides the ability to make units in one subsystem visible to units in another. This facilitates arranging the project in groups of related files that are more manageable.

Ada units may be made visible to other subsystems by means of an export/import set. These sets are created within a view, and may contain the entire contents of the view or only selected units. These imports effectively link the units of the export set to the importing view. As the units of the export set are updated, the changes are visible to the importing view.

The default export set includes all of the units within the view. Multiple export sets can be created within a view, but an importing view can only access one set from each exporting view. An

export set may be imported into another view in another subsystem, but not into a view within the same subsystem.

Apex does allow views to import each other, but this should be approached with caution. This option allows interdependencies among the groups of Ada units, and conflicts with the concept of segregating the files by related functions.

CMVC

The configuration management and version control (CMVC) utility is used to limit file access within views. CMVC changes the system attributes of the files under its control so that only Apex may make any modifications to the files. In addition, CMVC guarantees that each file may only be changed or updated by one user or process at a time.

Once CMVC accepts the control of a file, CMVC changes the system attributes of the file so that it becomes read only to everyone except Apex. The only way to modify, copy, or delete these files is to use the file maintenance commands within Apex. This control assures that the files will not be haphazardly changed or deleted from outside of Apex.

The CMVC further controls access to files by requiring that a file must be "checked out" before any modifications may be made to it. If the file is checked out by a user, no other user or process may modify the file. When the file is checked back in, it is once again available for check out, and is assigned a new version number. A new version is only created when the file is checked in. A file may be checked out, modified and saved repeatedly, and checked back in, but a new version is only recorded at the point of check in; all of the intermediate changes are lost.

The CMVC keeps track of all previous versions of each file, and these versions may be accessed for various operations. With Object Tool: Set Version, the active copy of the file may be selected from all of its past versions. Apex uses the active copy in compilation and linking with other units. This function allows further development of a unit while an older version is still being used for interface with other units.

The CMVC will compare two versions of the same file and show the differences between them. There are two ways to find the differences between versions. Within the Set Version options, highlighting two consecutive versions and selecting Control: Show: Differences will bring up a display window with the file and the differences. Within the Directory Viewer, highlight the file to be examined and select Control: Show: Differences. This will bring up a dialogue box in which any two versions of the file may be selected for comparison. The window that opens as a result of either of these commands has the file and all of the differences between the two versions highlighted. Lines contained in the second version and not in the first will be highlighted in blue. Lines contained in the first version and deleted from the second will be highlighted in red.

Rational Rose/Ada Interaction

Rational Rose/Ada is an object oriented design diagramming tool for use with Ada programs. Within Apex, Rose/Ada offers the option to build a module diagram for the Ada system being

developed. This automatic generation of diagrams can be very helpful in the review of a system. The discussion of Rational Rose/Ada will be limited to the module diagrams that are produced through the Apex reverse engineer function.

When working with a large project, a graphical representation of the relationships between units of code is often helpful in understanding and developing the project. This representation is often called a module diagram. Apex has the capability to generate a module diagrams of its subsystems and views.

After the files have been put into views, a command from the Directory Viewer menu is required: Tools: Reverse Engineer. A dialogue box will appear and will prompt for the objects or views to be engineered. The option "Include closure of views/units" will cause the diagram to include all views and subsystems required for coding. The dialogue box will also prompt for the name of the petal file to be generated. This file is read by Rose/Ada to generate the diagrams, and should have the extension ".ptl".

After exiting Apex, Rational Rose/Ada should be run. Under the File menu, Import is the correct choice. The dialogue box that appears prompts for a petal file, and offers the ability to navigate the directory structure to obtain the file. When the petal file and "OK" are selected, Rose/Ada reads the petal file. To view the diagrams, Browse: Module Diagram must be selected from the main menu. The dialogue box that appears will have a list of subsystems, usually including the subsystem that was to be diagrammed as well as the "lrm.ss" and "predefined.ss" as the Unix system default base. The diagrams are saved as model files, and may be transferred to the DOS based Rational Rose/Ada with no changes.

To illustrate the progression of the module diagrams, an Ada version of the Corps of Engineers database program DB1383A will be used. All of the required units were loaded into one view in a Apex subsystem. When Apex generates the module diagrams, it does not put the files in any particular order. The file symbols are arranged in a diagonal line across the page so that the dependency arrows are aligned (Figure 1). This representation is not very useful, so some rearranging was attempted. The first attempt to group the symbols by function causes a confusing result (Figure 2).

A different approach was needed. The second attempt began by breaking DB1383A into separate subsystems by topic: TUI, Textual User Interface screen generation packages distributed by AdaSoft, Inc.; Admin, the administrative and driving packages; Entup, the interface to the database packages; and Report, the report generating packages. The view importing options within the Apex system were used to connect the subsystems to the required supporting packages. A subsystem module diagram was then generated through Apex and loaded into Rose/Ada. The symbols of the diagram were then rearranged to a legible pattern, as shown in Figure 3. Following this, each of the views were reverse engineered, and their petal files imported into Rose/Ada. Some rearranging was necessary, but the results were easily readable, as shown in the included figures.

Conclusion

The Apex system is easy to work with. The interface is menu driven, and most required information can be selected from pick lists. The help facility is thorough in its coverage of topics.

The tutorial is designed to be worked in segments, and is connected directly to the help facility. The subsystem and view arrangements encourage an object oriented approach to design. File integration is handled automatically; the most difficult part is typing in the target view. The Rational Rose/Ada interface is point and click. This system is designed to make software projects more manageable, and Apex easily reaches this goal.

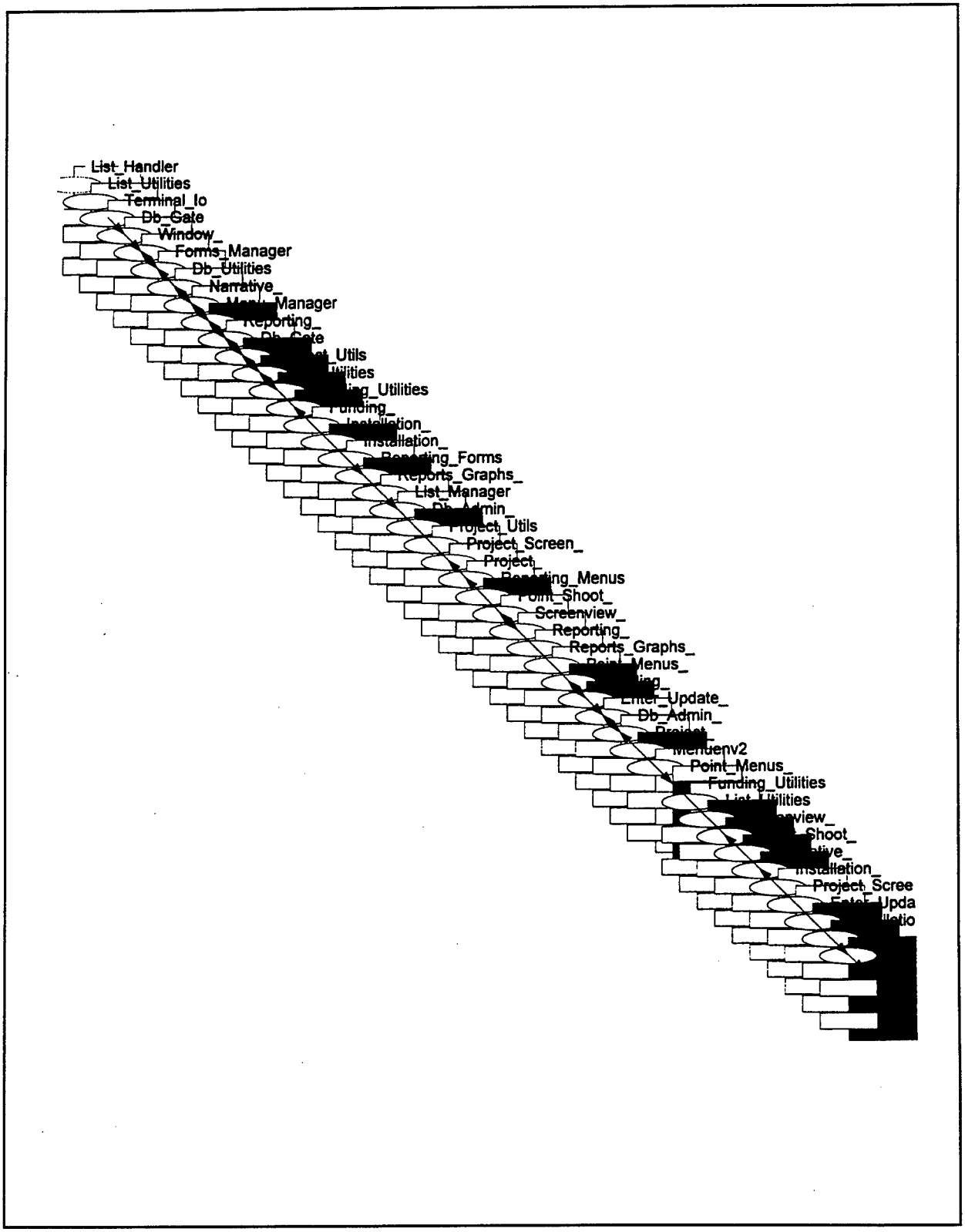


Figure 1. User generated modules for DB1383A

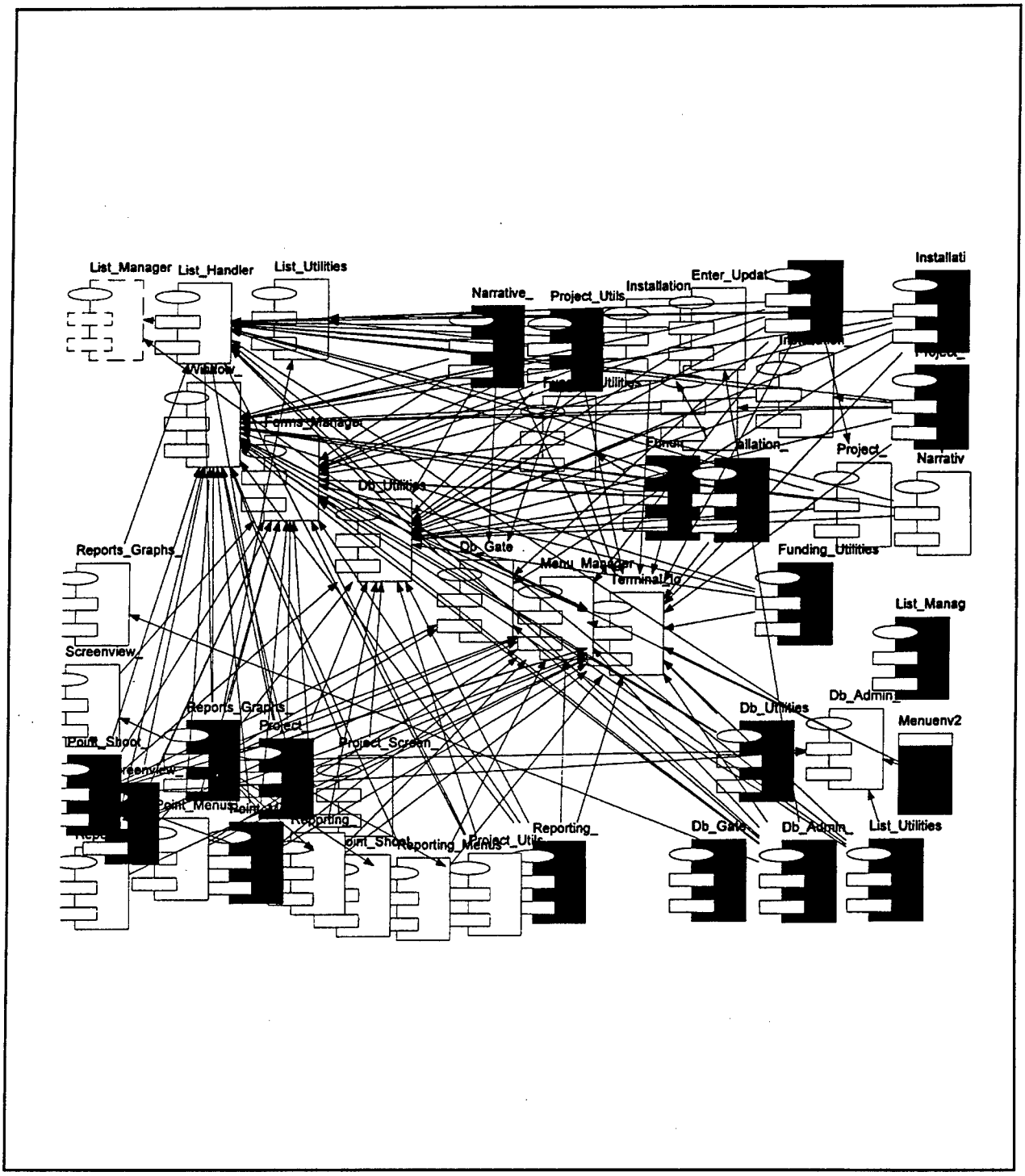


Figure 2. Relationships of user generated modules for DB1383A

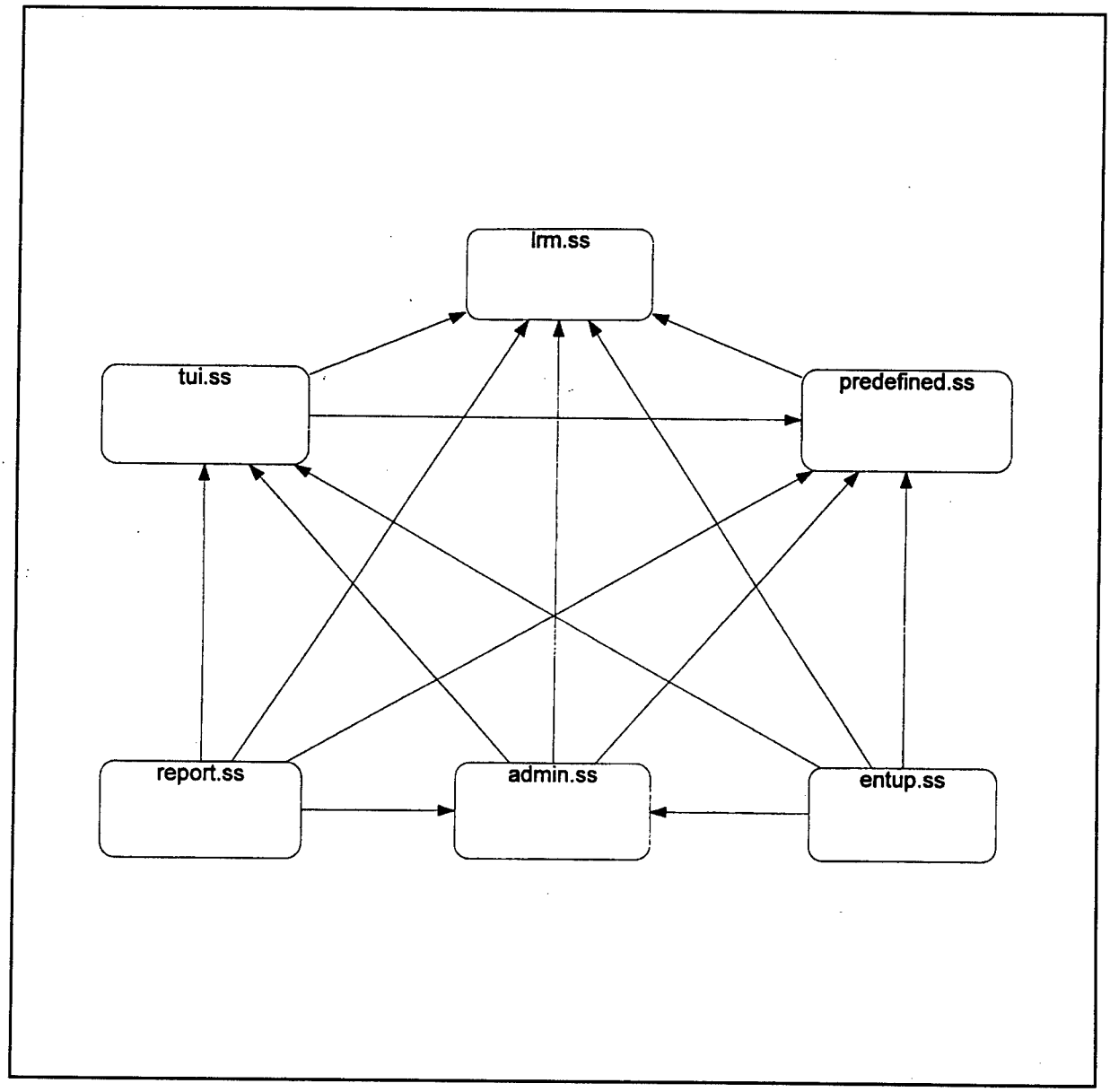


Figure 3. Subsystem relationships for DB1383A

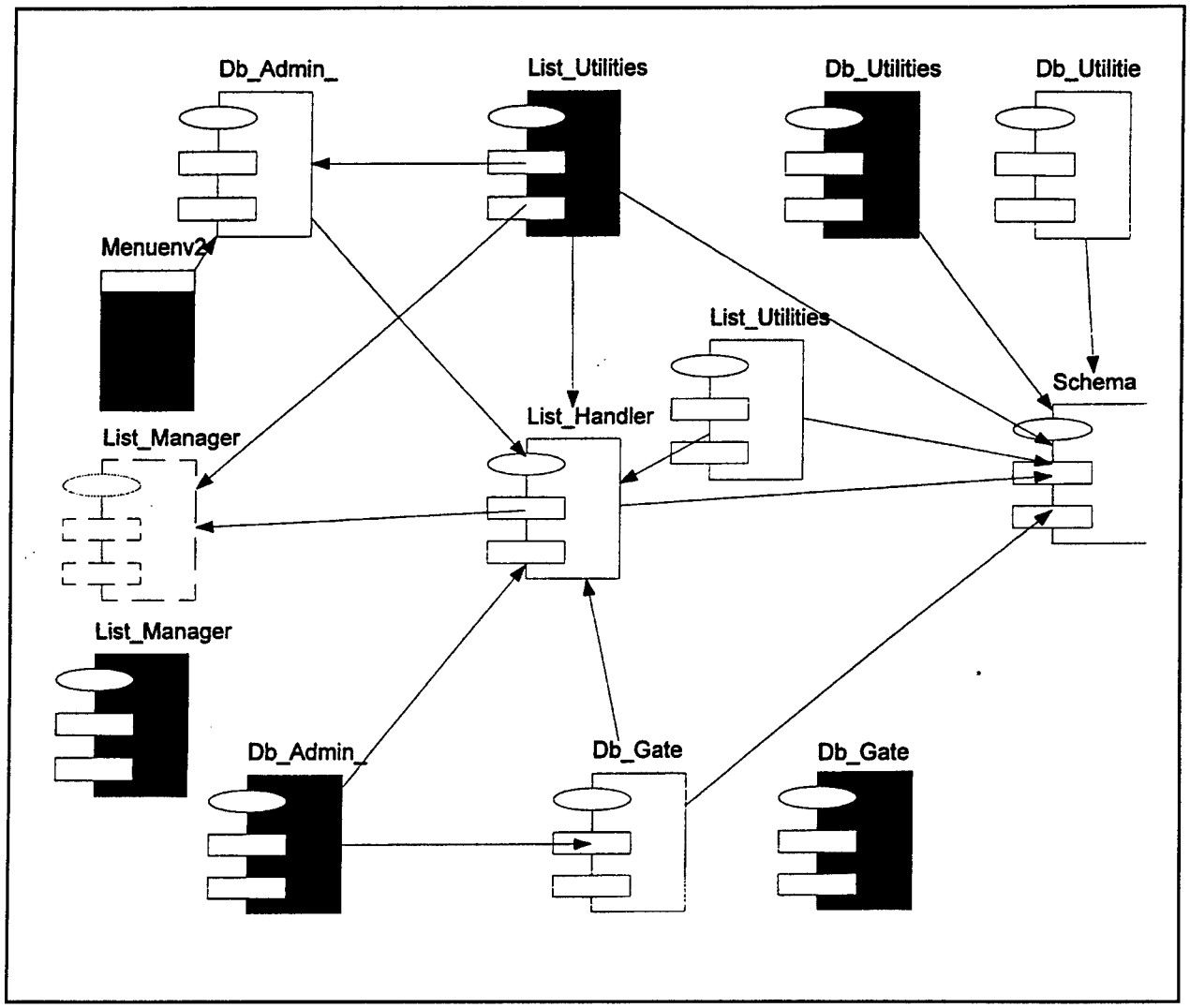


Figure 4. Relationships within the Admin subsystem for DB1383A

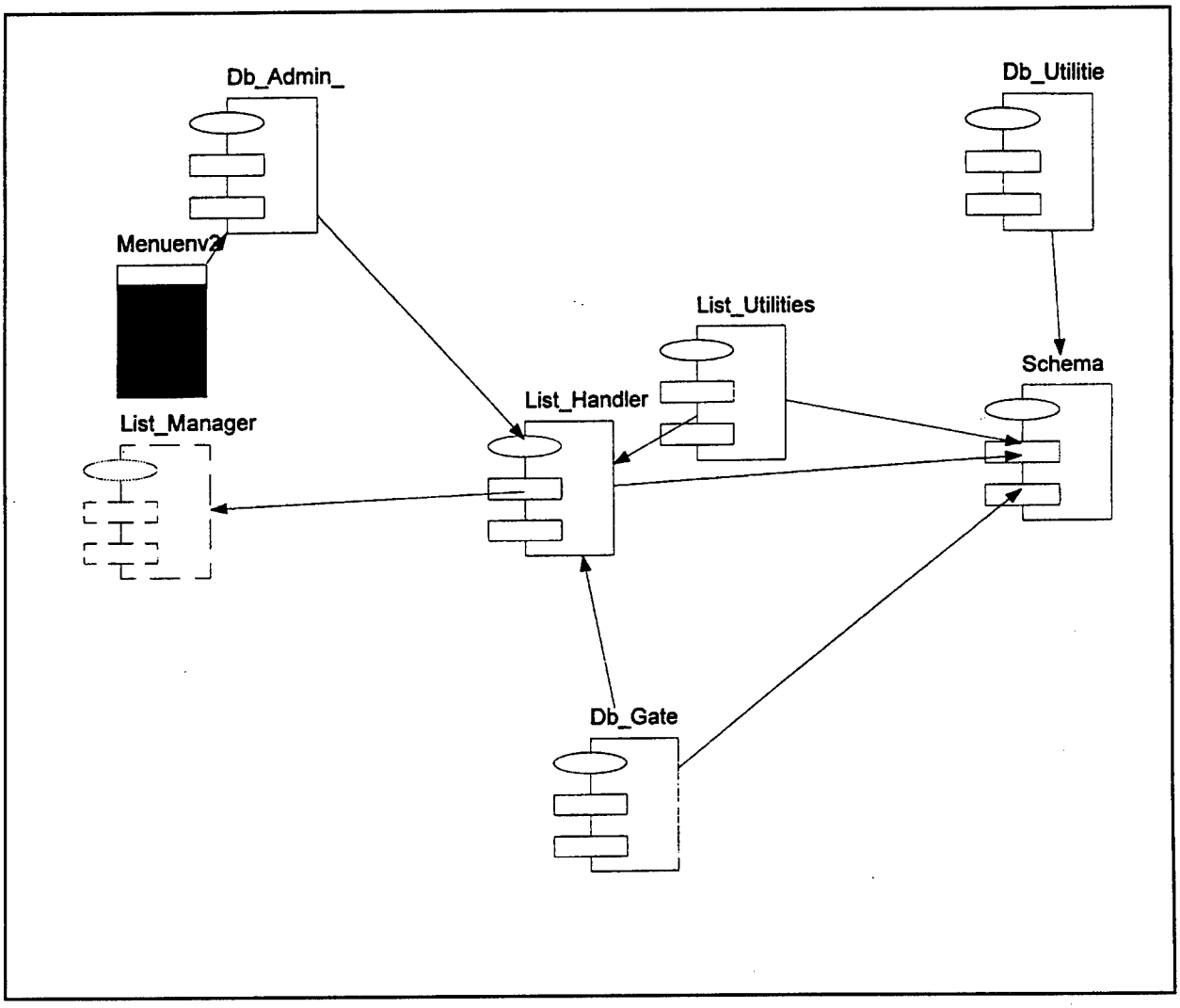


Figure 5. Relationships of specifications and driver within the Admin subsystem for DB1383A

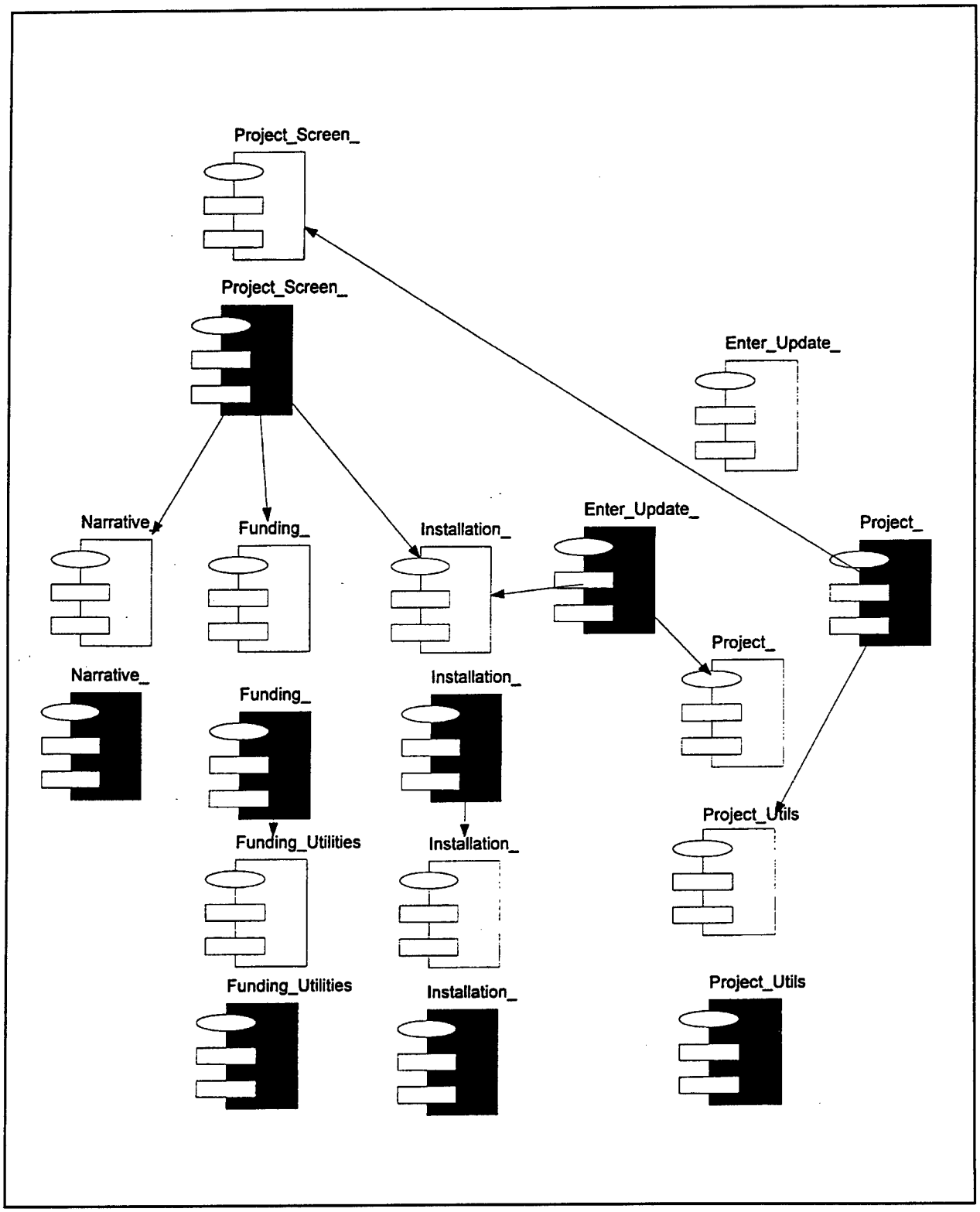


Figure 6. Relationships within the Entup subsystem for DB1383A

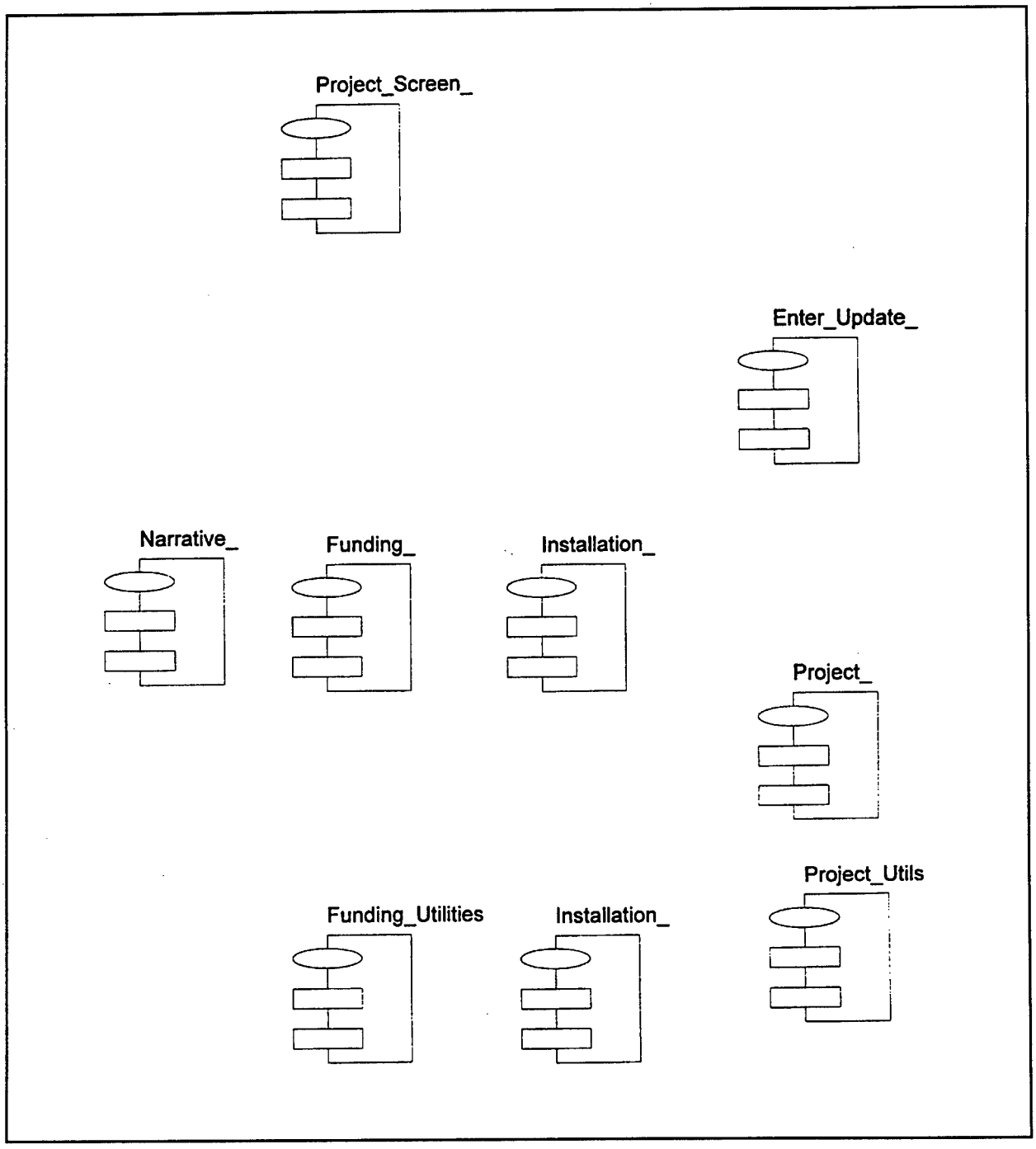


Figure 7. Specifications within the Entup subsystem for DB1383A

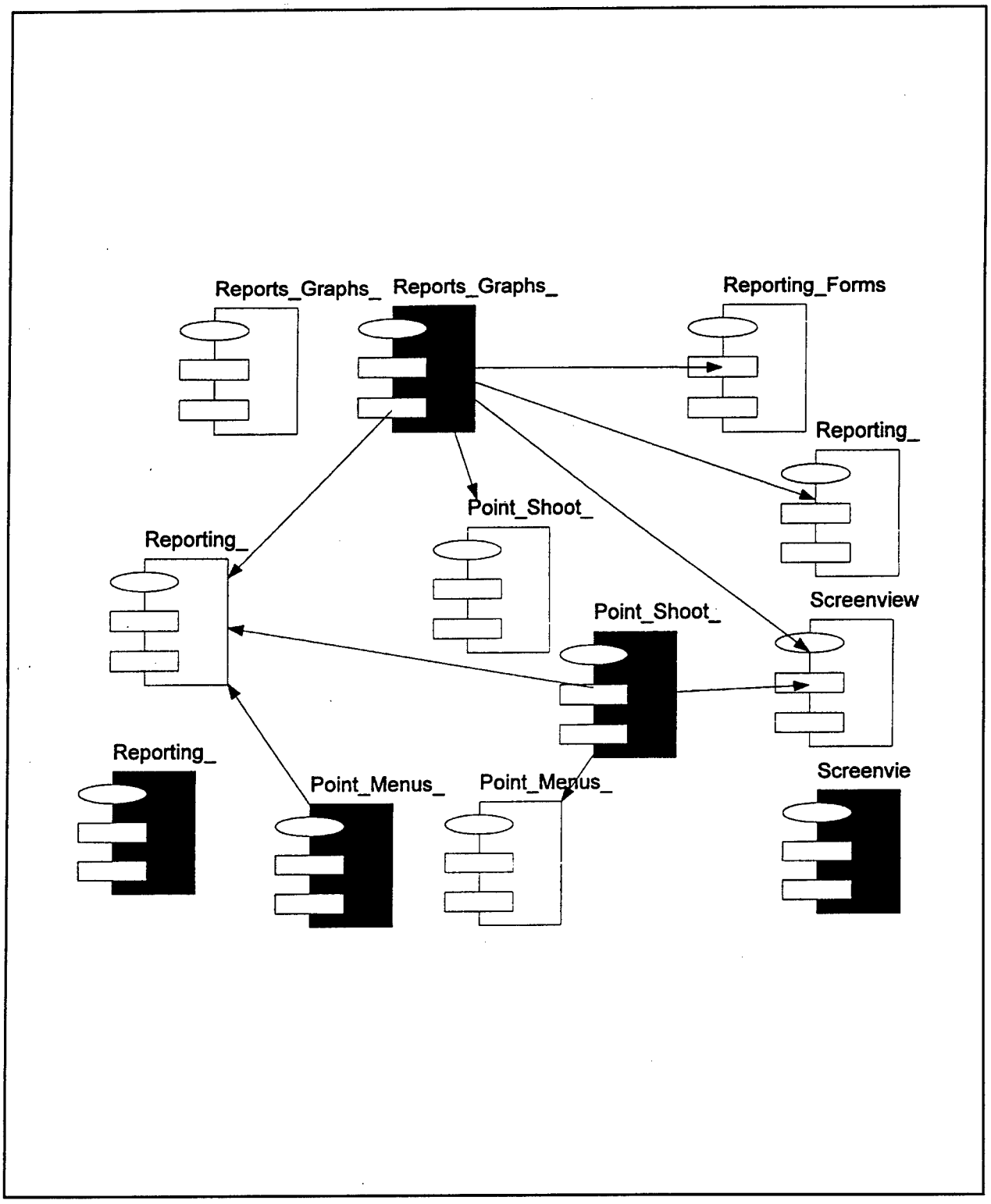


Figure 8. Relationships within the Report subsystem for DB1383A

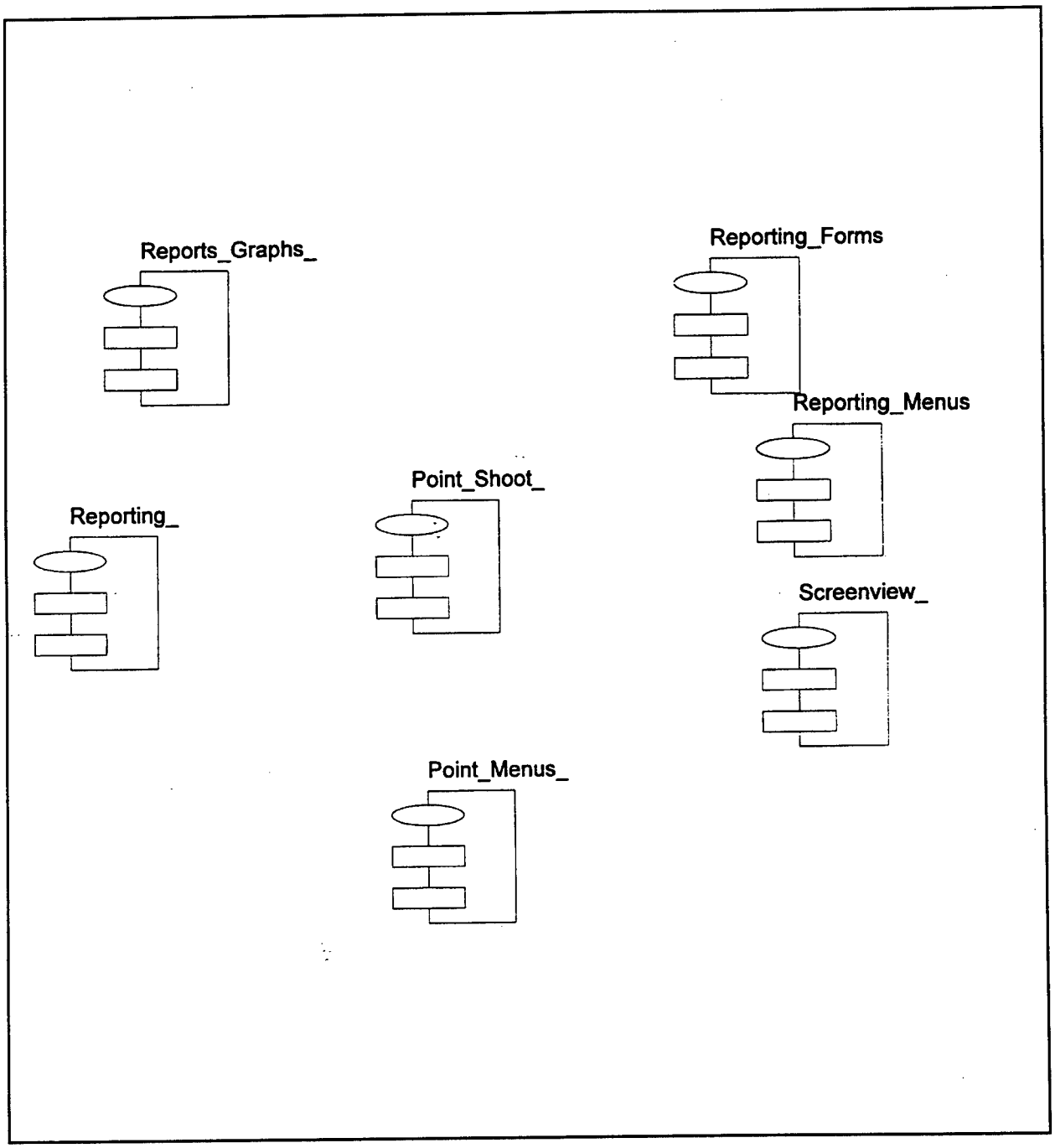


Figure 9. Specifications within the Report subsystem for DB1383A

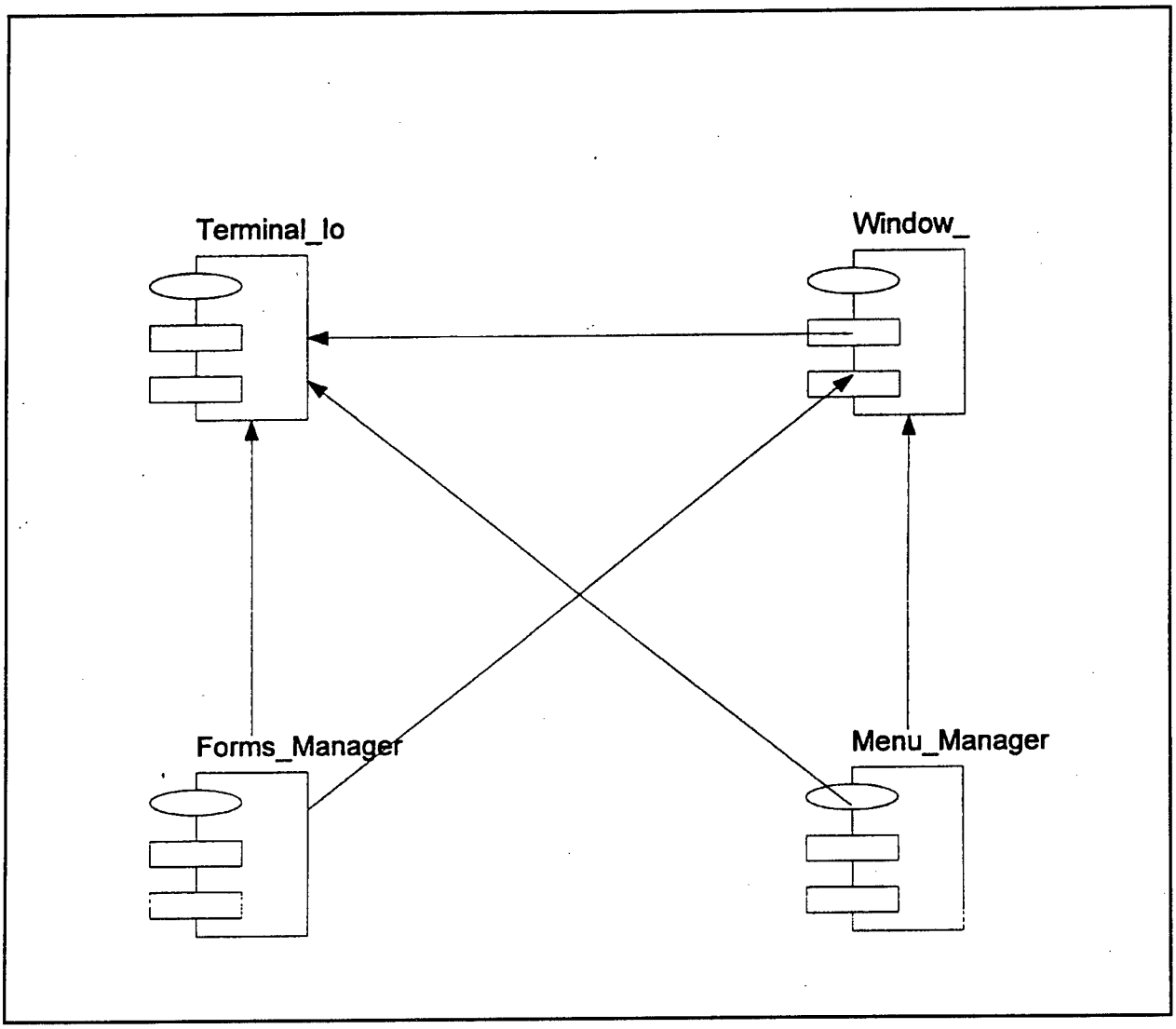


Figure 10. Relationships within the TUI subsystem from AdaSoft, Inc.

REPORT DOCUMENTATION PAGE

Form Approved
OMB No. 0704-0188

1. AGENCY USE ONLY *(Leave blank)*	2. REPORT DATE	3. REPORT TYPE AND DATES COVERED
	July 1996	Final report

4. TITLE AND SUBTITLE
APEX CM-VC and Reverse Engineering

5. FUNDING NUMBERS

6. AUTHOR(S)
Pamela M. Woof

7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)
University of Memphis
Memphis, TN 38152

8. PERFORMING ORGANIZATION REPORT NUMBER

9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)
U.S. Army Engineer Waterways Experiment Station
3909 Halls Ferry Road, Vicksburg, MS 39180-6199
US Army Corps of Engineers, Washington, DC 20314-1000

10. SPONSORING/MONITORING AGENCY REPORT NUMBER
Technical Report ITL-96-6

11. SUPPLEMENTARY NOTES
Available from National Technical Information Service, 5285 Port Royal Road, Springfield, VA 22161.

12a. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution is unlimited.

12b. DISTRIBUTION CODE

13. ABSTRACT ***(Maximum 200 words)***

Apex is an Ada development environment. This environment contains tools for developing large software projects, including file management and control tools. Apex is designed to be a multiuser environment, and includes file safeguards to lend stability to the development. The environment also allows several versions of any Ada unit to be held in reserve, to be used in the future.

Apex divides the work are into subsytems that contain views. Views are the file access mechanisms that coordinate the Ada files. All development work is performed within views. There are several different kinds of views with differing levels of file management options and restrictions.

The configuration management and version control (CMVC) utility is used to limit file access within view constraints. CMVC changes the system attributes of the files under its control so that only Apex may make any modifications to the files. In addition, CMVC guarantees that each file may only be changed or updated by one user or process at a time. CMVC also keeps track of old versions of each file and makes these versions available for certain functions.

Rational Rose/Ada is an object oriented design diagramming tool for use with Ada programs. Within Apex, Rose/Ada offers the options to build a module diagram for the Ada system being developed.

(Continued)

14. SUBJECT TERMS
Ada
Apex
Configuration Management Version Control
Waterways Experiment Station

15. NUMBER OF PAGES
26

16. PRICE CODE

17. SECURITY CLASSIFICATION OF REPORT	18. SECURITY CLASSIFICATION OF THIS PAGE	19. SECURITY CLASSIFICATION OF ABSTRACT	20. LIMITATION OF ABSTRACT
UNCLASSIFIED	UNCLASSIFIED		

13. (Concluded).

This automatic generation of diagrams can be very helpful in the review of a system.

This report is not intended to serve as an introduction to all of Apex. Included are discussions of the view as a work space, importing foreign text files, importing between views, configuration management and version control, and the interaction between Apex and Rational Rose/Ada. The discussion of Rational Rose/Ada will be limited to the diagrams that are produced through the Apex reverse engineer function.